SOFTWARE FOR FAULT DIAGNOSIS IN COMBINATIONAL CIRCUITS

by

S. K. GANDHI

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

DECEMBER 1975

SOFTWARE FOR FAULT DIAGNOSIS IN COMBINATIONAL CIRCUITS

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

by

S. K. GANDHI

to the

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

DECEMBER 1975

CERTIFICATE

This is to certify that the thesis entitled, "SOFTWARE FOR FAULT DIAGNOSIS IN COMBINATIONAL CIRCUITS" is a record of work carried out under my supervision and that it has not been submitted elsewhere for a degree.

Dr. K.L. Kodandapani
Lecturer
Department of Electrical Engineering
Indian Institute of Technology
Kanpur

Kanpur
November 1975

ACKNOWLEDGEMENT

I am indebted to Prof. V. RAJARAMAN and Dr. K.L.K. KODANDAPANI for initiating the work presented and for encouraging me to complete the work successfully.

I thank Mr. M.S. Krishnamoorthy for teaching me the basic concepts of software architecture and IBM-1800 ASSEMBLY language.

Special mention should be made about Mr. B. SRINIVASAN, Mr. WARYAM SINGH and in particular, Mr. G.K. RAO for their help during the different phases of the work.

My thanks are also due to Mr. J.S. RAWAT for the excellent typing work done.

December 1975
Kanpur

S.K. GANDHI

AM I JUST A PIECE OF SOFTWARE

Am I just a piece of software?
Being programmed and run through
The Universe of Hardware.

Am I just a non-subscripted variable?
Incremented, indexed and inialised.
Or, one element in an array?
Subroutined, stored and zeroised.
Input with information
To be stored until later.
Output on examination
All the compiled data
Debugged, Down-time, Bucket Overflow
Jammed-up Off-line, Micro-Macro.

I am just a piece of software
Being programmed and run through
The Universe of Hardware.

-Alec Gill

SYNOPSIS

The combinational logic circuits are prone to faults of the stuck-at type. An optimal diagnostic table (DT) is necessary to detect the faults occurring in such circuits. The DT serves as an input to an automatic test system (ATS) which conducts fault detection experiments on the unit under test.

A FORTRAN based system called SYSTEM I is implemented on IBM-1800 computer. SYSTEM I is capable of generating the complete test set for a combinational circuit consisting of NAND and NOR gates and is found to work satisfactorily. The study of the design of a fault-diagnosing system is also attempted in this thesis.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Faults in Digital Systems:

Digital systems have revolutionized the fields of communication, control and computers. The systems intended to carry out the different information processing tasks are complex and sophisticated. The performance requirements imposed on these systems are so stringent that developing methods to ensure their continuous and safe operations become vital.

Physical devices and components constituents of these systems are prone to failures or faults which may cause the system to malfunction. Faults in digital systems can be classified into several categories according to the underlying physical causes. Aging or manufacturing defects may cause the component to deteriorate giving rise to 'marginal faults'. Noise and close tolerances can cause 'intermittent' faults which appear only during some intervals of time. Faults which originally are intermittent may eventually become 'solid'. The faulty condition which does not change with time is referred to as a solid or permanent fault. A permanently open collector or base lead of a transistor is a solid fault. Under these conditions the

output from the transistor may be either stuck-at-low or stuck-at-high. The faults of this type which cause a change in the logical behaviour of the circuit are known as 'logical faults' where we are least interested in the physical cause of the occurred fault but on the type of the fault being either stuck-at-0 (s-a-0) or stuck-at-1 (s-a-1). Logical faults are the most common in digital logic circuits and most of the efforts in fault diagnosis are oriented towards developing procedures for detecting logical faults of the stuck-at-0 or stuck-at-1 type.

1.2 Schemes of Fault Detection and Diagnosis:

Depending upon the underlying principle, error detecting schemes can be classified as Hardware schemes[3] or Error diagnosing Software schemes[4] . After detecting the error the system can do one of the following:

1. Signal the user that the data being processed may contain errors or conversely assure that the system is operating correctly.

2. Initiate the restarting of an operation interrupted by an error.

3. Provide the maintenance personnel with information about the machine status.

4. To check the subsystems periodically to detect any faults; if fault is detected either to reconfigure itself so that it functions normally or to indicate the location of the occurred fault. Such systems are known as self-checking systems and are used where it may be impossible to keep the maintenance crew at the site of the equipment.

Of the several functions described above the first two are performed by hardware and the others by special software routines. In general error detecting schemes involve both type of schemes to achieve as much of the above functions as allowed by cost versus high performance trade-off. Both schemes have their own merits and demerits. A comparison[3] of the two schemes is given in Table 1.1

Table 1.1

Comparison of error detecting schemes

Program	Hardware
1. Cost of Software and periodically running them.	Cost of designing and building error detection circuits into the system.
2. Error is pinpointed to the period between successive running of error detection routines.	Error is indicated as and when it occurs.
3. Intermittent errors are difficult to detect.	Intermittent errors can be detected by the same H/W.
4. Operates periodically	Operates continuously.

In large systems like digital computers both type of schemes are implemented. For example,

(i) Parity error detecting logic - H/W

(ii) Error detection routines - S/W

But in small digital systems the cost of providing a built-in error detecting logic may be as high as that of the system itself. To test small systems and IC chips with large number of internal gates, a separate Automatic Test System (ATS)[4] may be constructed and this system can be supplied with the required test information generated from a 'Fault Dictionary Generator' as that developed in this thesis.

1.3 Fault Diagnostic Table:

A 'test' T_k of length s is defined to be a sequence of input vectors $X_{k1} \ldots X_{ks}$ to be applied to the unit under test (UUT) together with the corresponding primary output sequence. It is written as,

$$T_k = \{X_{k1} \ldots, X_{ks};\ Z_{k1} \ldots, Z_{ks}\} \quad \text{where}$$

$$X_{kj} = \langle x_1 \ldots x_i \ldots x_n \rangle ;\ x_i \in \{0,1\} ;\ j \in \{1,2,\ldots,s\}$$

and

$$Z_{kj} = \langle z_1 \ldots z_i \ldots z_n \rangle ;\ z_i \in \{0,1\} ;\ j \in \{1,2,\ldots s.\}.$$

Qualitatively Z_{kj} is the correct primary output when the input vector to the system is X_{kj}. If the system is faulty

internally and the output sequence corresponding to the input sequence $X_{k1}\ldots,X_{ks}$ to be $Z_{k1}\ldots,Z'_{kj}\ldots,Z_{ks}$ and $Z_{kj} \neq Z'_{kj}$; $\exists_j \in \{1,2\ldots s\}$, then we say that T_k detects the fault. In other words T_k detects the fault in the system iff,

$$Z_{kj} \oplus Z'_{kj} = 1; \quad j \in \{1,2\ldots,s\} \quad \text{where} \ldots \qquad (1.1)$$

Z_{kj} is the output vector when the fault is present and the input to the circuit being X_{kj}.

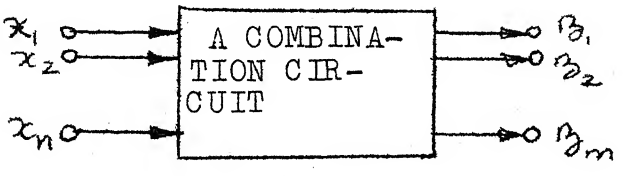

Fig. 1.1

The tests are classified as,

1. Fault Detection Tests and
2. Fault Diagnosis Tests.

Fault detection tests when applied to the UUT will be able to say whether the specified class of fault(s) have occurred or not. These tests do not try to carry any information as to the location of the fault. That is, the 'diagnostic resolution' of these tests is zero. Whenever it is possible diagnostic tests aid in locating the faults. It should be noted that not all possible type of faults can be located by a test set. So we say that a test or a

combination of tests can locate a specified class of faults. Normally the length of diagnostic tests is more as compared to the tests of the other type because of the increased information carried by the former class of tests. It is possible to show through examples that a fault 'f' may be detected by a number of tests and a test T_k may detect a number of faults. A proper exploitation of the above property helps to diagnose the fault.

If in the UUT a single line is faulty then such faults are called single faults and simultaneous more than one line faults are known as multiple faults. Multiple faults are difficult to detect and we do not consider these faults in this work. Suppose the UUT contains r lines. Then there are 2r possible single stuck-at faults. The faults and the test sequence corresponding to each fault can be represented in a tabular form, called a Diagnostic Table or Fault Table. It is in the form of a binary matrix. A 1 or 0 entry in the matrix indicates that the test associated with the row can or cannot detect the fault or fault equivalence class of the column respectively.

In Example 1.1 we find out the DT of a NOR gate using the definition of fault tests.

EXAMPLE 1.1

To determine the diagnostic table of a NOR gate.

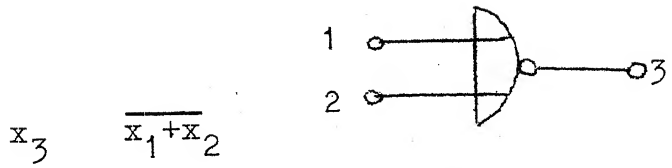

Fig. 1.2

Table 1.2

x_1	x_2	x_3	x_1	x_2	x_3	x_1	x_2	x_3	x_1	x_2	x_3
0	0	1	1	0	0	0	1	0	0	0	0
0	1	0	1	1	0	0	1	0	0	1	0
1	0	0	1	0	0	1	1	0	1	0	0
1	1	0	1	1	0	1	1	0	1	1	0
	(a)			(b)			(c)			(d)	

Table 1.2(a) gives the truth-table of a NOR gate under fault free conditions. Assume that s-a-1 fault occurs in line 1. The output x_3 is given in Table 1.2(b). Similarly (c) and (d) give the truth tables when lines 2 and 3 are s-a-1 and s-a-0 respectively.

From Table (a) & (b) we can find that the input vector 00 when applied to the gate will detect line 1 s-a-1, written as 1/1, where the first integer represents the line number and the second the type of fault. It can also be verified that 00 can detect the faults 2/1 and 3/0. By

applying 00, we will be able to detect any one of the faults 1/1, 2/1 or 3/0. But the faults are indistinguishable when we observe the response of line 3 to 00. Otherwise we say that they form an 'equivalence class'. If with respect to a primary output, n faults have the same test sequence, then they can be grouped in a set. This fault equivalence has created a need for introducing additional test points in any digital circuit so that all the faults are distinguishable.

The complete DT for the gate can be easily derived from the truth tables and is given below:

Table 1.3

Diagnostic Table

	FAULTS				
TESTS	A	B	C	D	E
00	1	0	0	0	0
10	0	1	0	1	0
01	0	0	1	1	0
11	0	0	0	1	0

Fault Equivalence Classes/Faults

A 1/1, 2/1, 3/0

B 1/0

C 2/0

D 3/1

E fault-free gate.

Sometimes the column associated with the fault-free gate (E) contains the correct response of the output of the circuit for the applied test input. For fault-detection the DT may contain some redundant tests. If we remove the test 11 from the DT, but all the faults detected by it, are still detectable by either 10 or 01. However for fault location, in this case, all the tests are required.

1.4 Procedures for System Testing and Diagnosis:

Once the DT is generated for a circuit, there should be some systematic ways of testing or diagnosing it. The procedures usually employed are called 'Test Procedures' and there are two kinds of testing methods: sequential and combinational. In a sequential test procedure the jth test input vector is determined from the responses elicited in the j-1 previous inputs. In a combinational procedure the input is specified regardless of the order in which it is applied. Either kind of procedures can be applied to either kind of digital systems.

1.4.1 Sequential Procedures: Consider the DT shown in Table 1.3. For each test input the faults can be classified into two partitions depending upon the output responses, being either good or bad. In Table 1.4 the faults are

grouped into two partitions. This is a modified form of Table 1.3. Table 1.4 is the starting point for the sequential procedure.

Table 1.4

Tests	Response	
	0	1
00	B,C,D,E	A
01	A,C,E	B,D,
10	A,B,E,	C,D
11	A,B,C,E	D

0 : good m/c response

1 : bad m/c response.

Assume fault B is present. When we apply 00 it is unable to detect it. As far as 00 is concerned the circuit (m/c) is good. But 01 will detect the fault B. This is shown in the above table. In an actual fault test generation procedure such tables are obtained by 'Fault-Simulation' techniques [6] .

Sequential procedure essentially aims at narrowing the the partitions into partitions of smaller size, till it ends up with a situation where no further narrowing down is possible. When a partition of size 1 is achieved, a single

fault would have been located. Sequential procedures can be represented by a diagnostic tree.

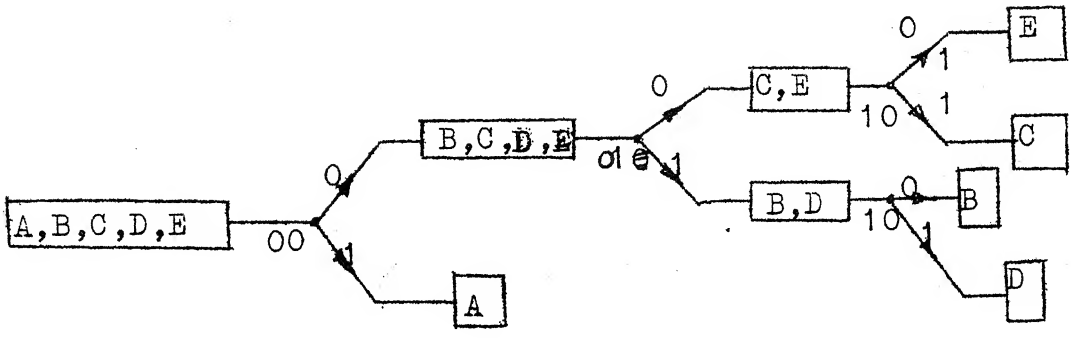

Fig. 1.3 A Diagnostic Tree

Each node of the tree represents a decision point where we apply the input shown as a binary string near the node to the UUT and observe the response. Depending on good or bad machine responses we branch along the paths. Suppose the fault is E. We would have applied the input sequence 00,01,10 and got the output sequence as 0,0,0. If for the same input sequence we get the output sequence as 0,1,1 then the fault is D. This can be checked by tracing along paths in the tree. Sequential procedures in general take more memory and less time.

1.4.2 Combinational Procedures: All the input tests are applied in order and the corresponding observed outputs are stored in a vector. Operating this vector with the normal

correct outputs using an Exclusive OR operator gives a resultant vector. This is compared with each column of the DT. If a match occurs the fault associated with that column is said to have occurred. Combination procedures take more time and less memory. If in the process of applying a test to the UUT we find a single 1 in any row and if that test shows bad machine response, then the testing can be stopped at that instant. We would have isolated a fault at that point. Such an early stopping of the procedure is known as 'early termination'.

1.5 Aims of the Thesis:

The theory and methods of fault diagnosis are studied. Efficient and easily implementable algorithm(s) for generating the DT and diagnosing combinational digital circuits consisting of NAND and NOR gates, are chosen. Single faults are assumed The multiple fault detection theory is under development. A system for generating the DT is designed and implemented on IBM-1800. The design of a fault diagnosis System is also presented. The software routines developed will be used for the design of an Automatic Test System. The size of the digital system is limited by the core memory of the computer. The circuit can have 99 lines, 8 primary input and 8 primary output lines. The number of reconvergent fan-out lines in the circuit is restricted to 15. However can be increased, by changing the

proper parameter in the system. IBM-1800 was chosen for implementation because of its ON-LINE I/O facilities. The Software developed should help to locate faults using the 1800 computer if suitable H/W interface is provided.

1.6 Outlines of the Thesis:

In Chapter 2 the fundamental concepts and techniques underlying fault detection and diagnosis are presented along with relevant theory. Boolean Differential Calculus is introduced and its application to the generation of DT is also studied. Chapter 3 consists of the stages involved in the design of softwares for DT generation and fault detection. In Chapter 4 we present the results obtained and the comparison of this system with systems already implemented elsewhere. The drawbacks of the system developed is also discussed along with suggestions for improvement.

CHAPTER 2

THEORY AND METHODS OF FAULT DIAGNOSIS

In this chapter the theoretical principles involved in the design of algorithms for fault diagnosis are presented. In order to have a physical insight into the problem some simple techniques are discussed. The trend in fault diagnosis is to use the Boolean Difference (BD) more and more. Certain properties of Boolean Difference make it easier to develop algorithms. Some popular algorithms are compared and one which is easy to implement and efficient is presented in detail. Towards the end , fault table minimisation techniques and fault location methods are discussed. Wherever necessary examples are provided so that the techniques can be easily understood.

2.1 Truth-Table Method:

The most obvious method of obtaining the fault table is by comparing the truth-tables of the normal and faulty circuits.

Consider a combinational circuit with m primary output and n primary input lines. The primary output z_i is written as,

$$z_i = f_i(\underline{X}) \text{ where}$$

$$z_i \in \{0,1\}$$
$$\underline{X} = \langle x_1 \ldots x_i \ldots x_n \rangle \; ; \; x_i \in \{0,1\}.$$

For any set of faults F and any fault $\alpha \in F$, let

$$z_i = f_i(\underline{X})$$

be the ith output when α is present. An input vector $\underline{X}^j = \langle x_1^j \ldots x_i^j \ldots x_n^j \rangle$ is a test for detecting the fault if and only if,

$$f_i(\underline{X}^j) \oplus f_i^{\alpha}(\underline{X}^j) = 1; \; \exists i \in \{1,2 \ldots m\} \quad \ldots(2.1)$$

In other words $z_i \oplus z_i^{\alpha}$ gives the test set for detecting the fault α. In Example 1.1 this method was used to construct the DT of a NOR gate. This is the most inefficient method for we have to construct the truth-table each time when we find out the test set of a fault.

2.2 Fundamentals of Boolean Differential Calculus:

Definition 2.1: [8,14]

As before let

$$z_i = f_i(\underline{X})$$

and $\underline{X} = \langle x_1 \ldots x_j \ldots x_n \rangle$

be the primary output and primary input respectively. The Boolean Difference (BD) of z_i with respect to any input line x_j (denoted by dz_i/dx_j) is given by

$$\frac{dz_i}{dx_j} = f_i(x_1, x_2, \ldots, 0, \ldots, x_n) \oplus f_i(x_1, x_2, \ldots 1, \ldots x_n) \quad \ldots\ldots (2.2)$$

If $\frac{dz_i}{dx_j} = 0$, this implies that z_i is independent of x_j. In otherwords a change in x_j from 0 to 1 or vice versa does not cause a change in z_i. In a practical situation line x_j may be redundant with respect of z_i. dz_i/dx_j in general may be a function of x_k's, $k \neq j$. The value of z_i will depend upon the value of x_j if and only if the remaining variables assume values such that $dz_i/dx_j = 1$. This condition, we shall prove later, is one of the necessary conditions so that a fault in line x_j can be detected. If we want to determine the BD of the Primary Output Line (POL) with respect to any one of the internal lines of a combinational digital circuit, then we express the POL as a function of its Primary Input Lines (PIL) and the internal lines. Then the definition could be extended.

Certain theorems can be proved to relate the BD of complex functions to the BD of simple functions. These theorems can be easily derived from the Difinition 2.1 and fundamental principles of Boolean Algebra.

Theorem 2.1:

$$\frac{d\bar{z}}{dx_j} = \frac{dz}{dx_j}$$

Theorem 2.2:

$$\frac{dz}{d\bar{x}_j} = \frac{dz}{dx_j}$$

Theorem 2.3:

$$\frac{d}{dx_i}\left(\frac{dz}{dx_j}\right) = \frac{d}{dx_j}\left(\frac{dz}{dx_i}\right)$$

Theorem 2.4:

$$\frac{d(z.y)}{dx_i} = z\,\frac{dy}{dx_i} \oplus \frac{dz}{dx_i} \cdot \frac{dy}{dx_i} \oplus y\,\frac{dz}{dx_i}$$

Theorem 2.5:

$$\frac{d(z+y)}{dx_i} = \bar{z}\,\frac{dy}{dx_i} \oplus \bar{y}\,\frac{dz}{dx_i} \oplus \frac{dy}{dx_i} \cdot \frac{dz}{dx_i}$$

Theorem 2.6:

$$\frac{d(z \oplus y)}{dx_i} = \frac{dz}{dx_i} \oplus \frac{dy}{dx_i}$$

Proofs for these theorems are available in Ref.[12]

EXAMPLE 2.1:

Calculate the BD of the POL with respect to the PILs of the circuit in Fig. 2.1

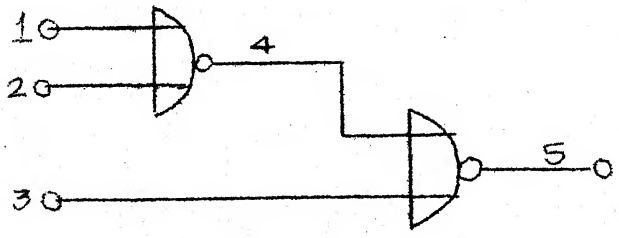

Fig. 2.1

Solution:

$$x_5 = \overline{\overline{x_1 \cdot \bar{x}_2} + x_3}$$

$$x_5 \ (x1 = 0) = 0$$

$$x_5 \ (x_1 = 1) = \overline{\bar{x}_2 + x_3} = x_1 \cdot \bar{x}_3$$

$$\frac{dx_5}{dx_1} = 0 \oplus x_2 \cdot \bar{x}_3 = x_2 \cdot \bar{x}_3 .$$

Similarly,

$$\frac{dx_5}{dx_2} = x_1 \cdot \bar{x}_3 \quad \text{and}$$

$$\frac{dx_5}{dx_2} = \overline{x_1 \cdot \bar{x}_2}$$

If we apply at the input terminals the combination-10 then any change in the value of x_1 can be propagated to x_5. This is known as 'sensitizing'.

2.3 Method of Path Sensitizing:

The test which detects a given fault is an input combination which produces an incorrect output when the fault is present. The necessary and sufficient conditions for a test to detect a given fault are:

(1) The signal value at the site of the fault should be opposite to that caused by the fault.

(2) Any change of signal value at the site of the fault should cause a change of at least one output of the circuit.

Condition (2) can be satisfied by making the BD of the output with respect to the faulty line to be 1. In that case the path from the location of the fault to the output line is said to be 'sensitized'.

In the general procedure for fault-test generation using the path sensitizing technique, the faulty line is assigned a value opposite to the fault condition. A value of 1 is assigned for a s-a-0 fault and a 0 for s-a-1 fault. The logical values of other PILs are so chosen that we sensitize a path from the fault to the output line. The inputs to the gate feeding the faulty line are also set to proper values so that condition is met. We trace back till we assign values to all PILs.

EXAMPLE 2.2:

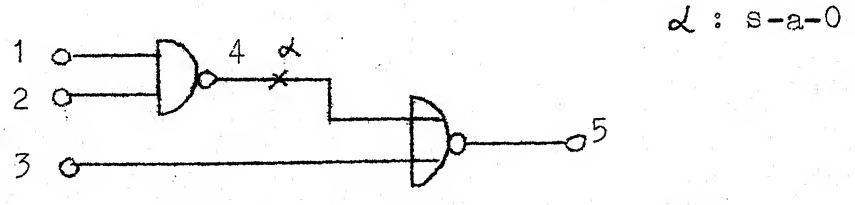

Fig. 2.2

Assume s-a-0 fault in line 4. The path 4-5 can be sensitized by setting line 3 to be 1. We assign a value of 1 to line 4 (since the fault is a s-a-0). This would require line 1 to be 0 or line 2 to be 0 or both. The test set for detecting line 4 s-a-0 is given by

$$\{001, 011, 101\}$$

The path sensitizing procedure attempts to sensitize only one path at a time. If the circuit contains reconvergent fan-out lines, the procedure may not generate the correct test set. In some cases, it may not be possible to sensitize all the paths from the faulty line to the POL. Example 2.3 gives the pit falls of the path sensitizing technique.

EXAMPLE 2.3:

In the circuit of Fig. 2.3 let us try to derive the test set for detecting the fault α s-a-0 by sensitizing the path ABH.

1.

We set $x_1=1$, $x_2=x_3=0$ so as to apply a signal opposite to the faulty value to fault α and propagate the effect of the fault through the gate A. Keeping $x_4 = 0$ propagates it through B. The condition for sensitizing the

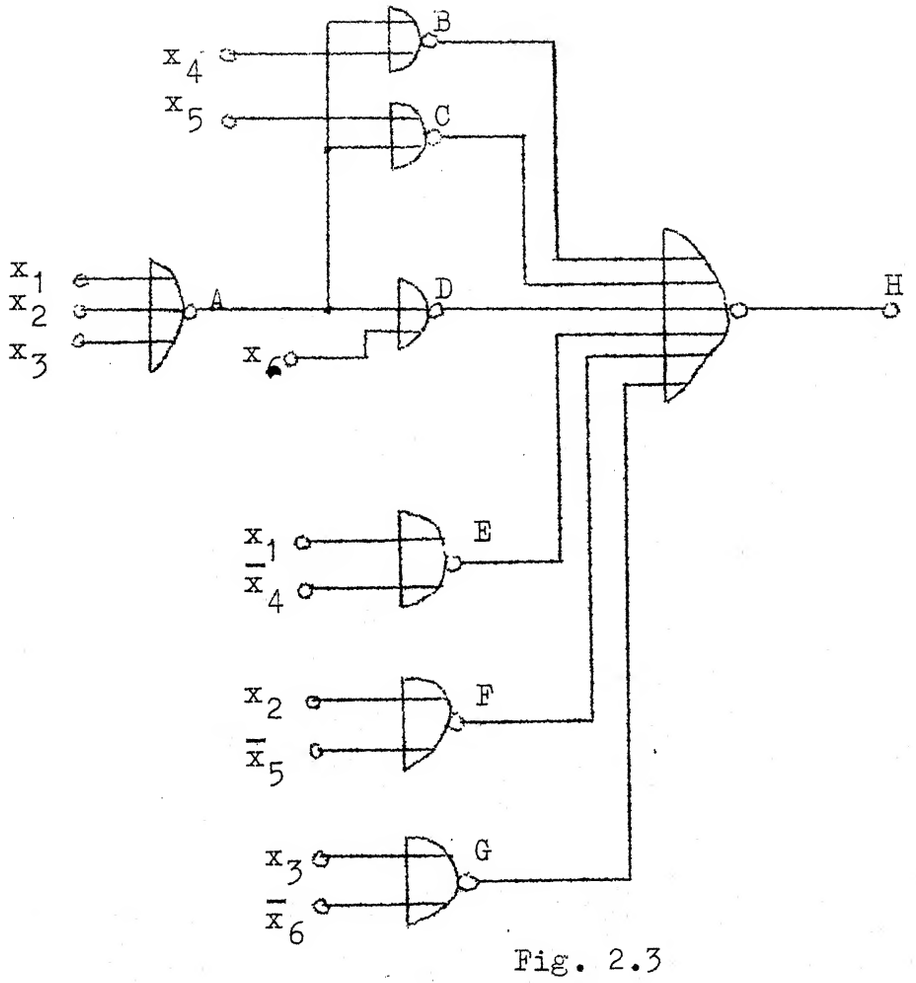

Fig. 2.3

path ABH is, C=D=E=F=F=0. C and D can be made 0 by setting $x_5=x_6=1$. But this would cause G and F to be 1 and resulting in a conflict. Similarly it can be seen that it is impossible to sensitize the single paths ACH or ADH. It is noted that $x_4=x_5=x_6=0$ sensitizes the three paths simultaneously and also makes E=F=Q=0. The test for is then 100000.

From the above example, we find a necessity for sensitizing more than one paths. There are algorithms like D-algorithm, which identify all the paths and sensitize them.

2.4 Method of Boolean Differences:

Consider a combinational circuit with a POL z and internal lines $U = \{x_1, x_2, \ldots x_i, \ldots x_n\}$. The primary input vector

$$\underline{X} = \langle x_1 \ldots x_j \ldots x_m \rangle \quad \text{where } m \leq n$$

$$x_j \in U.$$

The test for detecting a s-a-0 fault in any line $x_i \in U$ is given by

$$x_i \cdot \frac{dz}{dx_i} \qquad \ldots\ (2.3)$$

and for detecting a s-a-1 fault in x_i by

$$\bar{x}_i \cdot \frac{dz}{dx_i} \cdot \qquad \ldots\ (2.4)$$

Eqns. 2.3 and 2.4 can be derived as follows:

Let T_α be the complete test set for detecting α s-a-0 in line $x_i \in U$. T_α is given by

$$T_\alpha = z \oplus z^\alpha$$

Now,

$$z = f(\underline{X}).$$

(or) $$z = F(\underline{X}, x_i)$$

For x_i s-a-0,

$$z^{\alpha} = F(\underline{X}, 0)$$

$$T_{\alpha} = z \oplus F(\underline{X},0)$$

$$= x_i F(\underline{X},x_i=1) \oplus \bar{x}_i F(\underline{X},x_i=0) \oplus F(\underline{X},0) \quad [\text{Ref:12}]$$

$$= x_i F(\underline{X},1) \oplus F(\underline{X},0).(\bar{x}_i \ominus 1)$$

$$= x_i F(\underline{X},1) \oplus F(\underline{X},0).x_i$$

$$= x_i (F(\underline{X},1) \oplus F(\underline{X},0))$$

$$= x_i \cdot \frac{dF(\underline{X},x_i)}{dx_i}$$

$$T_{\alpha} = x_i \cdot \frac{dz}{dx_i}$$

The other formula can also be proved in a similar manner. The first term in the test gives the condition for which the signal value at the site of the fault is opposite to that of the fault. The second term specifies the condition for sensitizing the path.

2.5 The D-Algorithm:

D-algorithm is a systematic version of the path - sensitising technique, which tries to sensitize more than one path simultaneously.

Definition 2.2:

Let symbol D be defined to represent a signal that is 1 in the normal circuit and 0 in the faulty circuit. The

symbol $\bar{D}$ will be used when the signal takes values in the order other than that taken by D. Table 2.1 gives few logical operations on D and $\bar{D}$.

Table 2.1

+	0	1	D	$\bar{D}$
0	0	1	D	$\bar{D}$
1	1	1	1	1
D	D	1	D	1
$\bar{D}$	$\bar{D}$	1	1	$\bar{D}$

.	0	1	D	$\bar{D}$
0	0	0	0	0
1	0	1	D	$\bar{D}$
D	0	D	D	0
$\bar{D}$	0	$\bar{D}$	0	$\bar{D}$

In the algorithm, a function of n variable $f(x_1, x_2 \ldots, x_n)$ is represented by the singular cover of the function [2]. The 'don't care' values are shown by x. Two types of inputs to a gate are possible. In the first type, the inputs are those which cause the output of a gate to be different from the normal output when the fault is present. These inputs are represented by the 'primitive D-cubes' of the gate. The next type consists of the 'propagation D-cubes' of the gate, which cause the output of a gate to depend only on one or more of its specified inputs and hence to propagate a fault on these inputs to the output.

The primitive D-cubes of any fault in a logic gate can be obtained from the singular covers of the normal and

faulty gates. The singular cover of the normal gate is partitioned into two classes a_0 and a_1 depending upon the output of the gate being either 0 or 1 respectively. A similar partition can be done on the singular cover of the faulty gate. Call the partitions b_0 & b_1. Perform the following operations:

(1) The members of a_1 are intersected with members of b_0 excepting the output coordinates. The nonzero (nonempty) intersections correspond to those inputs which cause the output coordinate to be 1 in the normal circuit and 0 in the faulty one. The output vertices of these cubes are assigned D.

(2) The members of a_0 and b_1 are intersected. Nonzero (nonempty) intersections give the inputs which cause the output of the circuit to be 0 in the normal and 1 in the faulty circuits. A $\bar{D}$ is assigned to the output vertices, of these cubes.

Primitive D-cubes of gates with multiple outputs are computed by dividing the singular covers of the normal and faulty gates into partitions a & b according to their outputs. Subscripts are used with a and b corresponding to the outputs. The cubes are obtained by intersecting members of a with members b with different subscripts. The output

coordinates of those nonempty intersections are assigned values. If both members of the intersection have the same output, then that is assigned as it is, to the intersection. If an output coordinate in 'a' has a 0 and in 'b' has a 1, then that coordinate in the intersection is assigned a $\bar{D}$. We assign a D if the output coordinate in 'a' has a 1 and 0 in 'b'.

The propagation D-cubes are also calculated for each of the gate in the circuit. Because the propagation D-cubes of a gate is different for its different inputs, for each input it is evaluated. For any input i of a gate,

(1). Partition the singular covers into a_0 and a_1 according to the output coordinates.

(2). For each element in $a_0(a_1)$, complement the ith input if it is not x and the output. We obtain sets $b_1(b_0)$.

(3). Intersect the elements of partitions with the same subscripts. The nonempty intersections give the conditions for which a change in the ith input is propagated to the output of the gate.

(4). For each cube obtained assign a D to the ith input and D or $\bar{D}$ to the output coordinate depending on whether or not the value of the output coordinate in the cube is the same as the ith input value.

The same input variable of a gate may have more than one propagation D-cubes. (example: Exclusive- OR gate)

EXAMPLE 2.4:

Determine the primitive and propagation D-cubes of the circuit shown in Fig. 2.4.

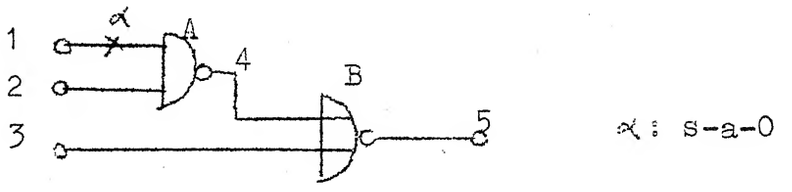

Fig. 2.4

Assume x s-a-0 in line 1. The singular cover for gate A of the normal circuit is given by

1	2	4	
1	1	0	a_0
0	x	1	a_1
x	0	1	a_1

With x present, the singular cover is given by

1	2	4	
1	1	1	b_1
0	x	1	b_1
x	0	1	b_1

The partitions made on the singular covers are also shown. Intersecting a_0 with b_1 and ignoring the output

coordinate we get the primitive D-cube of gate A as 11xDx→.

The singular cover of gate B is

3	4	5	
1	x	0	a_0
x	1	0	
0	0	1	a_1

Let us calculate the propagation D-cube of B for its input 4. Complementing the coordinates of 4 and 5 in the cubes a_0 and a_1 we get b_{01} and b_0 respectively.

3	4	5	
x	0	1	b_1
0	1	0	b_0

We get XXODD̄ by intersecting b_0 and a_0 and assigning proper D value. The cube will be obtained when we intersect the other two partitions namely b_1 and a_1.

The procedure to determine the test of the circuit consists of two stages. In step 1, we intersect the primitive D-cubes of the faulty gate with those of the gates at a higher level till we reach the output line. This is known as 'D-drive'. This is equivalent to sensitizing a path from the fault to the output. The next step is called the 'consistency opn'. The D-cube obtained in step 1 is

intersected with the singular cover of the circuit. This operation yields the necessary inputs so that the fault is detected. The steps to be executed are:

(1). Obtain the singular cover and the propagation D-cubes of the circuit.

(2). Each test cube obtained in the D-drive is represented by t_c and a subscript denoting the step at which it is obtained. An activity vector is also assigned to each test cube. The activity vector gives the number of the gate upto which the sensitized path has been extended.

(3). An initial test cube t_c^0 is chosen from the primitive D-cubes of the fault under consideration. It is intersected with the propagation D-cube of one of the members of the activity vector of t_c^0 so that t_c^0 and the D-cube with which it is intersected have a D or $\bar{D}$ in at least one common coordinate. The intersection is the new test cube which is then labelled as t_c'. The new activity vector is determined and the procedure is repeated.

(4). The consistency operation is carried out by intersecting the test cube successively with the cubes in the singular cover until a sufficient number of circuit inputs for generating the signal values specified in the test cube have been assigned.

EXAMPLE 2.5:

To generate the fault test for the circuit shown in Fig. 2.4. (Example 2.4).

The primitive and propagation D-cubes were calculated as 11xDx and xxODD̄ respectively. Intersecting both we get 110DD̄ as the test cube. 110 is the fault test for line 1 s-a-0. The presence of D̄ as the output coordinate indicates that the normal value of the circuit is 0 and the faulty value is 1 when 110 is applied to the circuit.

Historically D-Algorithm presents a novel method of fault test generation. But at times, it becomes too tedious to use the method. If the circuit contains RFO lines and if there are more paths to the output from the fault then a lot of backtracking and duplication of certain efforts become imperative. New algorithms are available in the literature [13,9] which are capable of generating the test set in a straight forward manner.

2.6 The Yau and Tang's Algorithm:

Yau and Tang's Algorithm (YTA)[13] is based on the BD concept and BD chain. This algorithm is capable of generating the complete test set for any combinational circuit. This algorithm uses simple logical operations AND and OR, and operations on sets are not necessary as is the case with the D-algorithm.

Theorem 2.6:

Let z be a logic function realized by a combinational logic circuit. Let j be an internal line and i be another line in the circuit, such that every path from line i to z passes through j. Then,

$$\frac{dz}{dx_i} = \frac{dz}{dx_j} \cdot \frac{dx_j}{dx_i} \qquad \ldots\ldots (2.5)$$

where x_i and x_j are the variables representing line i and j respectively.

Proof:

Let $\underline{X} = \langle x_1, x_2 \ldots, x_n \rangle$ be the primary input vector. When all the path from i to z passes through j, we can write,

$$x_j = f(\underline{X}, x_i) \qquad \ldots\ldots (2.6)$$

and

$$z = F(\underline{X}, x_j)$$
$$= F(\underline{X}, f(\underline{X}, x_i)) \qquad \ldots\ldots (2.7)$$

We can have three cases.

Case i: Assume $\frac{df}{dx_j} = 1$ and $\frac{dx_j}{dx_i} = 1$. Then,

$$\frac{dz}{dx_i} = F(\underline{X}, F(\underline{X}, 0)) \oplus F(\underline{X}, f(\underline{X}, 1))$$
$$= F(\underline{X}, \zeta) \oplus F(\underline{X}, \bar{\zeta}) \quad \left[\because \frac{dx_j}{dx_i} = 1\right]$$
$$= 1 \quad \left[\because \frac{df}{dx_j} = 1\right]$$

Case ii: Assume $\frac{dx_l}{dx_i} = 0$. Then,

$$\frac{dx}{dx_i} = F(\underline{X}, f(\underline{X},0)) \oplus F(\underline{X}, f(\underline{X},1))$$

$$= F(\underline{X}, \delta) \oplus F(\underline{X}, \delta)$$

$$= 0$$

Case iii: Assume $\frac{df}{dx_j} = 0$, Then,

$$\frac{dz}{dx_i} = F(\underline{X}, \delta_1) \oplus F(\underline{X}, \delta_2)$$

$$= 0$$

Hence we have proved the theorem. This theorem is useful in evaluating the BD of the POL with respect to an input variable of a gate from the BD with respect to the output of the gate and the type of gate.

Algorithm:

1. All the lines in the circuit are numbered from the lower level to the higher level. Let this be

$$U = \{1,2,\ldots j \ldots l\}.$$

2. The logic function of all the lines and its complement are computed.

$\forall_j \in U$, X_j and $\bar{X}_j$ are expressed as a function of $\underline{X}$.

3. Let S be the set of RFO lines in the circuit. For every $j \in S$, $\frac{dx_l}{dx_j}$ is computed.

4. For every j = 1,1-1 ... 1 and $j \notin S$, the BD of line 1 $dx1/dx_j$ is calculated using Theorem 2.6 and the BD of the gate.

5. Calculate T_j using the following:

$$T_j^0 = X_j \cdot \frac{dx_1}{dx_j} \quad ; \forall j \in \{1,2 \ldots,1\}$$

$$T_j^1 = \overline{X}_j \cdot \frac{dx_1}{dx_j}$$

It is noted that the BD evaluation fails for the case of RFO lines if we use Theorem 2.6. Other methods are to be employed to do that function.

EXAMPLE 2.6

To calculate the test set for the lines of the shown circuit using YTA.

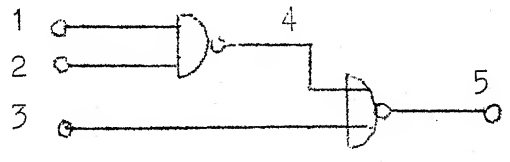

Fig. 2.5

Solution:

Step 1: Evaluating the functions.

$$x_4 = \overline{x_1 \cdot x_2}$$

$$\frac{dx_4}{dx_1} = x_2 \; ; \quad \frac{dx_4}{dx_2} = x_1$$

$$x_5 = \overline{x_3 + x_4}$$

$$\frac{dx_5}{dx_3} = \bar{x}_4 \; ; \quad \frac{dx_5}{dx_4} = \bar{x}_3$$

Step 2:

$$x_5 = \overline{\overline{x_1 \cdot x_2} + x_3} = x_1 \cdot x_2 \cdot \bar{x}_3$$

$$x_4 = \overline{x_1 \cdot x_2}$$

Step 3:

$$\frac{dx_5}{dx_5} = 1$$

$$\frac{dx_5}{dx_4} = \frac{dx_5}{dx_5} \cdot \frac{dx_5}{dx_4} = 1 \cdot \bar{x}_3 = \bar{x}_3$$

$$\frac{dx_5}{dx_3} = \bar{x}_4$$

$$\frac{dx_5}{dx_1} = \frac{dx_5}{dx_4} \cdot \frac{dx_4}{dx_1} = \bar{x}_3 \cdot x_2$$

$$\frac{dx_5}{dx_2} = \frac{dx_5}{dx_4} \cdot \frac{dx_4}{dx_2} \cdot x_1 \cdot \bar{x}_3$$

Step 4:

$$T_0^1 = x_1 x_2 \bar{x}_3 \; ; \; T_1^1 = \bar{x}_1 x_2 \bar{x}_3$$

$$T_0^2 = x_1 x_2 \bar{x}_3 \; ; \; T_1^2 = x_1 \bar{x}_2 \bar{x}_3$$

$$T_0^3 = x_3 \bar{x}_4 \; ; \; T_1^3 = \bar{x}_3 \bar{x}_4$$

$$T_0^4 = \bar{x}_3 x_4 \; ; \; T_1^4 = \bar{x}_3 \bar{x}_4$$

$$T_0^5 = x_1 x_2 \bar{x}_3 \; ; \; T_0^5 = \overline{\overline{x_1 \cdot x_2}} + x_3$$

Ref [13] presents a modified algorithm for evaluating the BD's if the available storage space is not sufficient for the computation. The algorithm may generate only a partial DT. YTA algorithm requires only simple operations. No necessity of back tracking has to be done. Only restriction on the algorithm is the core size which may eventually restrict the problem size. The DT generated is complete. This algorithm is selected because of its efficiency and simplicity to implement.

2.7 Fault-Table Minimisation:

Methods for generating the fault-table of a combinational digital circuit were discussed. The DT obtained may contain certain tests which are redundant. Depending upon the application of the DT being either fault detection or diagnosis, the DT may be minimised. For example in the case of fault-detection experiments, if a number of

tests detect a fault, it is sufficient to select a single test out of the possible tests. There are two ways of reducing the size of the DT. One is to use certain properties of the circuit under consideration to eliminate some duplication of effort during computation. Another is to use minimisation techniques once the DT is generated. Both are employed in any DT generation system.

Consider a NAND gate with n inputs $x_1, \ldots x_i, \ldots x_n$ and a output z.

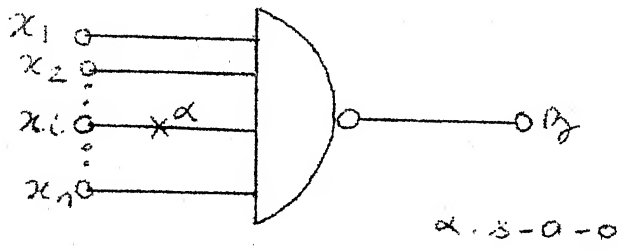

Fig. 2.6

The BD of z with respect to x_i is given by

$$\frac{dz}{dx_i} = x_1 \ldots x_{i-1} \ldots x_n \quad \ldots\ldots (2.6)$$

Suppose there is a fault s-a-0 in line x_i. The test for detecting the fault is given by

$$\frac{x_i dz}{dx_i} = x_1 \ldots x_i \ldots x_n$$
$$= \bar{z} \quad \ldots\ldots (2.7)$$

But $\bar{z}$ is nothing but the test that can detect a s-a-1 fault in the output of the gate. Besides $\bar{z}$ is independent of the value of i. In other words $\bar{z}$ is the test for detecting a s-a-0 fault in any one of the input lines and s-a-1 fault at the output. This is known as 'fault-equivalence' in a NAND gate.

In a similar manner in a NOR gate the s-a-1 faults at the inputs and s-a-0 fault at the output form an equivalents class. The elements of a fault-equivalence class (FEC) have the same test set.

While deriving the tests for s-a-0 faults at the inputs of the NAND gate, it is sufficient to check whether we have derived the tests for s-a-1 fault at the output. In the case of a NOR gate we derive only the tests for s-a-0 fault at the output and other faults which do not a form an equivalence class with the output s-a-0. The procedure described does not completely ensure that there are not more than one fault or fault equivalence classes having the same test set in the DT. If we have a NAND gate whose output is feeding one of the inputs of a NOR gate, the above procedure gives two FECs one for each of the gate. But it can be noted that the two FECs have the same test set. This would imply that it is essential to have a final minimisation stage wherein we reduce the table such that no two

faults or FECs have the same test set. On of the columns can be eliminated and the faults associated with it can be included with the faults of the other column. The DT arrived at till now can be used for either applications. Usually for diagnosis we leave the DT at this stage and do not try to minimise further.

For fault detection the DT can be minimised using one of the following procedures:

(1). Two columns of the DT are successively intersected. If the intersection is nonempty the two columns are replaced by the intersection. This process is repeated till we exhaust all the columns in the DT. Next we eliminate the rows. If a row contain 0's in all the position it can be eliminated. If any two rows have the same entry then one of them might be removed from the DT.

(2). In the DT if a row is covered by another row then it can be removed. If a column covers another column, then the covering column is removed.

2.8 Fault-Detection and Diagnosis:

Till now methods of DT generation and minimisation are studied. The ultimate aim of the DT is to use it for fault detection or location. Procedures suggested for diagnosis experiments will be discussed in this section.

Hayes [10] introduces a new method called 'Transition Counting (TC)' technique for fault detection in combinational circuits. In the conventional techniques the entire output string of the normal circuit corresponding to the input sequence is stored in memory. When the experiment is performed a bit by bit comparison is made with the stored output and the output of the UUT subjected to the experiment. If R is the length of the output string we would have to store $|R|$ bits in memory when $|R|$ is the length of the string R in bits. Instead of this, if we store the transition count (TC) of the correct output, it is required to store $\lceil \log_2 |R| \rceil$ bits only. The TC is nothing but the number of changes from 1 to 0 and vice versa starting from the left most bit to the right most bit. A special H/W can be introduced in the test equipment which gives the TC directly. To check for the presence of the fault we have to compare only the TC's of the correct output and the UUT's output.

Definition 2.3:

The TC C(R) of a string $R = r_1 \ldots r_i \ldots r_m$ is given by

$$C(R) = \sum_{i=1}^{m-1} r_i \oplus r_{i+1}$$

Theorem 2.7:

Let R be a m-bit binary sequence and $\bar{R}$ is obtained by complementing every bit of R.

(i) $C(R) = C(\bar{R})$ .... (2.8)

(ii) $0 \leq C(R) \leq m-1$ .... (2.9)

The proofs of the theorems follow straight away from the definition of C(R).

Definition 2.4:

Let $S = s_i \ldots s_m$ be a sequence of m-bit vectors applied to the UUT. S is said to be a TC test for UUT with respect to a fault in F if

$C(R_0) \neq C(R_i)$ for $1 \leq i \leq q$ where

$F = \{F_1, \ldots F_q\}$ and R_0 is the TC when S is applied to a normal circuit.

A fault in the UUT is said to be TC detectable if there exists some sequence S which is a TC test for the fault. Faults may exist which are conventionally detectable but may not be TC detectable. This is probable because $C(R) = C(\bar{R})$. Consider a s-a-0 fault at the output of a circuit. When we apply any test sequence we get C(R)=0. If that fault happened to be a s-a-1 fault again C(R) = 0. The two faults are TC undistinguishable but

detectable conventionally. The TC technique may not give full fault coverage. The coverage can be increased by orderingthe test sequence and applying them. Additional tests may be necessary for this ordering to be possible.

The use of a Decision Table as an input to an ATS was suggested by Scheff [17]. This is possible because of the equivalence between a diagnostic table and a decision table. Automatic programming methods [19] exist which can generate the code for running it in a system. Seshagiri [18] uses the probability or the failure rate of the faults in the circuit to determine the next test to be applied to the UUT. The diagnostic table is treated as the decision table. The test vectors in the DT correspond to the condition stub. The action stub is equivalent to the outcome of conducting a fault detection/diagnosis experiment. For fault - diagnosis we will be using a simple method which is essentially combinational procedure. All the tests are applied to the UUT and the resulting O/P sequence is stored. This when operated with the true response bit by bit using an Exclusive OR Operator gives us the 'signature' of the fault. All the tests corresponding to a '1' bit in the signature are intersected. Where ever there is 1 in the intersection the corresponding faults are identified. If during the EX'-OR operation we find the signature to be empty, then the UUT is fault free.

CHAPTER 3

THE PHILOSOPHY OF SYSTEM DESIGN

3.1 Introduction:

In this chapter an overall structure of an Automatic Test System (ATS) is described. The ATS is a total system capable of performing the following functions:

1. To verify the design of the circuit during the development stage. If an equipment designer has designed his digital system he would be interested in checking the designed one whether or not it conforms to the objective he had in his mind before initiating the design process. The verification of design can be done by a circuit 'Simulator'. There are different circuit simulators available, which can be integrated into the ATS.

2. To test modules developed during manufacture. Once the chips are manufactured before releasing, it should be checked for full performance within tolerance limits. We need to say whether the module has passed the test or not.

3. To diagnose during repair. A/C boards with large number of components may be a single module in a large equipment. The maintenance man will be interested in

knowing the exact component which has failed to function properly. The ATS should be able to locate the fault.

The Software of an ATS can be divided into two parts. The design of these parts is studied in this chapter.

3.2 Description of an ATS:

The block diagram[4] of an ATS is given in Fig. 3.1. Six modules constitute the structure of the ATS. They are:

1. Data base containing circuit topology
2. Circuit Simulation
3. Test Generation
4. A method of storing the DT
5. Automatic Test Equipment (ATE)
6. Fault Diagnosis

3.2.1 The Circuit Data Base: The efficient functioning of the ATS bears to certain extent on the coding procedure used to convey the circuit connection details to the system for further processing. To quote[4],

'The designer should know the data bases and its relationships to his available means of simulation and test generation and he should devise as efficient a means as possible of obtaining a simulatable model from the data base'.

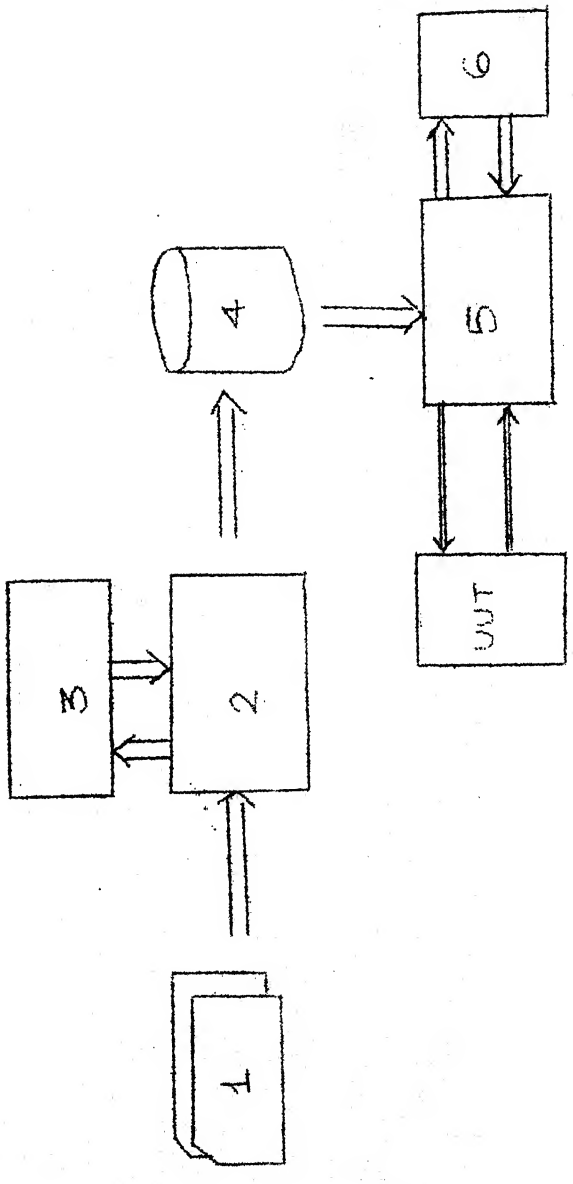

Fig 3.1 Structure of an ATS

3.2.2 Circuit Simulation: To generate the test set of the UUT, there should be available a model of the UUT. This is done by simulation. Another advantage of the simulator is that the designer can verify his design during development. The simulator should be capable of simulating the system behaviour from the circuit data base and the models of available components. The development in electronic technology is such that the basic building blocks of systems are complex and contain large gates. In that case the simulator should have the provision to accept new component models. Many aspects of a circuit can be simulated.

(1). Waveform network analysis

(2). Functional simulation

(3). Gate level simulation

(4). Layered simulation(a combination of 1 to 3)

Of the above type 1 is necessary only for design purposes. Type 3 is essential if we have TTL gates in our circuit. As the complexity of the components increases, functional simulation becomes more desirable than type 3. Layered simulation is a combination of all the three types and a particular aspect of a single component can be studied.

3.2.3 Test Generation: An optimal diagnostic table is the basic requirement of a test system. Ways of efficient test generation becomes a necessity if we want to use the system again and again. In general circuit simulator can be imbedded into the test generation for convenience. Test Generation systems can be implemented in any general purpose computer and only the test results are used by the ATE. Methods of fault test generation were described in Chapter 2.

3.2.4 Automatic Test Equipment: Once the test data is generated it can be stored in a disk file for permanent use by the ATE. If a number of components of the same logic, the ATE can be run the required number of times using the DT available in the disk. ATE should interpret the DT and apply test information to the UUT and observe the responses. ATE is a hardware set up which may have its own processors and instruction set.

3.2.5 Fault Diagnosis: There should be some way of interpreting the detection experiment conducted by the ATE H/W. For fault detection, the redundancy in the DT may be eliminated. All these functions are carried out by the fault diagnosis routine which is available as an integral part of the ATS.

In this thesis, the design of fault test generation is made along with the simulator and this is called SYSTEM I.

The design study of fault diagnosis module is also made. The software design is done by the systematic procedure given in Ref [23]

3.3 Design of SYSTEM I:

3.3.1 Format of Input: SYSTEM I is the fault-test generator of the ATS. It is capable of generating the DT of a combinational digital circuit. The combinational circuit can have 99 lines out of which at the most 8 primary input lines (PIL) and 8 primary output lines (POL) are permitted. The circuit may be containing NAND or NOR gates. Each of such gates can be fed by not more than 8 input lines. Reconvergent fan-out lines (RFO) are permitted to a maximum number of 15. However, this limitation can be relaxed by adjusting the suitable dimension in the system routines. The input to SYSTEM I has four different cards. They are:

1. No. of primary input lines card
2. No. of primary output lines card
3. Primary output lines card
4. Connection detail cards

The first three kinds of cards do not require an explanation. Connection detail cards contain information about the circuit topology. Information about a gate in the circuit can be supplied by a card. Fan-out points

(both RFO and non-RFO) are treated as a gate with single input and multiple outputs. The output(s) of a gate, the type of gate and the input(s) feeding the gate are specified in a card. The general format is given below:

O/P	CODE	1 ..INPUTS... 8

The code of the gate is in alphanumeric format. The whole card pack is followed by an 'end' card to indicate the end of data.

3.3.2 Specification of the Problem: Our aim is to design a system which can generate the DT of circuits whose specification is given in Sec. 3.3.1. The operations to be performed on the system are given below:

(1). To read input data cards and build CKT

(2). To evaluate the logical values of each line of the circuit as a function of PILs and build FUNC.

(3). For each POL of the circuit to enumerate paths from every PIL to the POL and build PATH

(4). To determine the RFO lines and to store them in FAN.

(5). To evaluate the boolean differences for the lines in the FAN and to build BLDF.

(6). To evaluate the boolean differences for the lines in the PATH other than those in FAN and to build BLDF.

(7). To identify the fault-equivalent classes (FEC) in the CKT with respect to the POL under consideration. To build the data bases FT and ECT.

(8). To compute the test set for every fault or FEC and to store the tests in FT.

(9). To minimise the FT for any redundancy.

(10). To output the FT with other required information for SYSTEM II.

The entire system is broken into 7 modules each of which does one or more of the above specified functions. Flow chart in Fig. 3.2 gives the sequence of the above operations and the modules performing those operations are also shown.

3.3.3 Data Structures: The complexity of a program to a large extent depends on the data structure of the components or variables used in the program. In this section we list the data structures used in SYSTEM I and in the next section we specify the format of the data structures.

1) CKT containing information about the input data.

2) EN containing all the POLs.

3) FUNC containing the logical values of the lines in the circuit as a function of PILs.

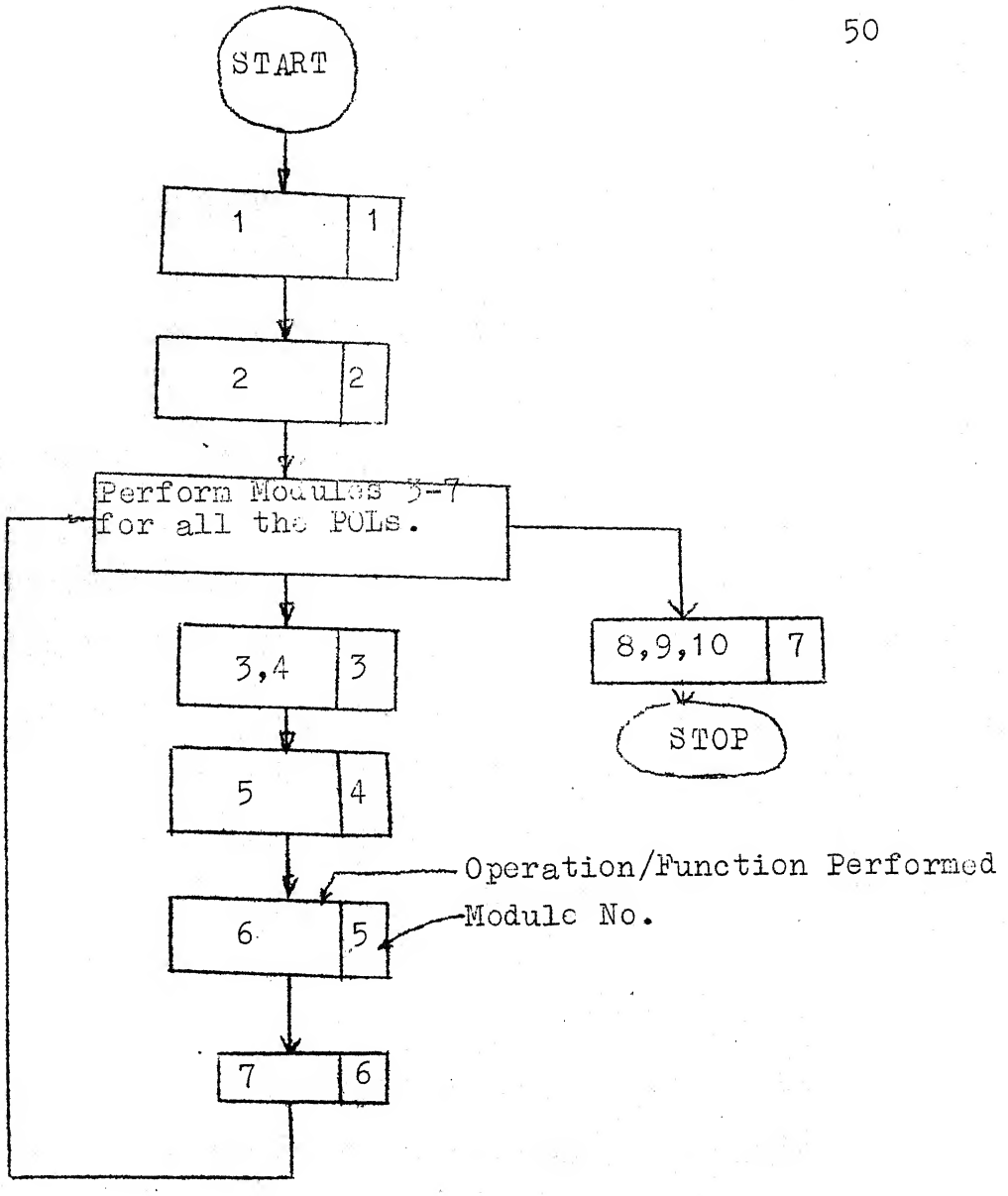

Fig. 3.2 Flow Chart of SYSTEM I

4) PATH containing all the paths from the PILs to the POL.

5) STACK as a working storage during the evaluation of the paths.

6) SP as a stack pointer.

7) MBIT the number of computer words required to store the logical value of a line in FUNC.

8) LBIT the number of bits less what the logical values of a line takes from MBIT.

9) FAN containing RFO lines.

10) STOR1, STOR2 as general purpose working storages.

11) BLDF containing the boolean differences (BD) of the lines in the circuit.

12) FLAG as a temporary storage to contain the faults.

13) FT the storage containing the fault-table or DT.

14) ECT containing the fault-equivalence classes in the circuit.

3.3.4 Format of the Data Structures: Of the above described data-structures only those which are important are studied in this section.

a) CKT: The format of CKT is given in Fig. 3.3. As there are not more than 99 lines in the circuit, 99 rows are allocated in CKT. Each row of the array corresponds to

line. The code of the gate and the inputs feeding the gate are available for each gate. The output line number of a gate is used as the key to access the gate information. The arrangement is sequential and in the ascending order of the keys.

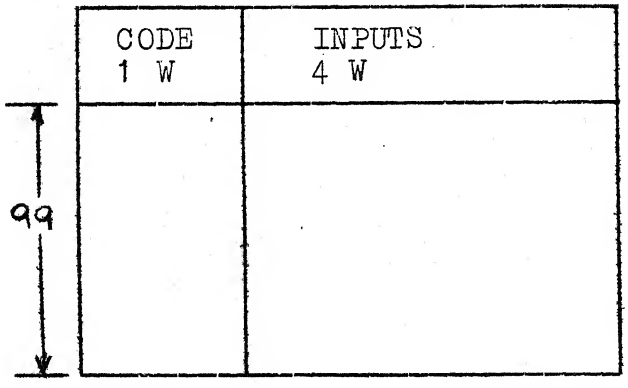

Fig. 3.3 The Format of CKT

Since each line of the circuit or UUT can be described by a 2 digit decimal number, we can pack 2 such lines into a word of the IBM-1800 computer. Assuming 8 inputs to a gate, we require 4 words to represent all the inputs. We allocate one more word to represent the code of the gate. Thus we need 5 words for each line of the circuit. The coding of the gates is done as follows:

1. The PILs are assigned the number 1.

2. Codes 2 and 3 are used to represent the fan-out lines and the RFO lines respectively.

3. Codes 4 and 5 are used to represent the NAND and NOR gates respectively.

(b) FUNC: In general, cubic notation is used to represent the boolean functions in a computer. Suppose there are r input variables. Then the logical values of any line can be represented by a set containing cubes with r coordinates or less depending upon the presence or absence of an input variable in that function. This would require the use of routines that perform cubic operations on these sets. Cubic operations are complex to perform and so we use another novel method which exploits the machine features of IBM 1800. Assume we have 8 input variables. Then there can be 2^8 input combinations possible. Any line of the UUT can have a subset of these 256 input combinations. Now a word of IBM 1800 has 16 bits. If we take 16 of 1800's words and assign the available 256 bits each to each of the 256 input combinations, we arrive at a positional system where the presence of a 1 or 0 in a word indicates whether the corresponding input combination is present in the function or not. The format is shown in Fig. 3.4. The additional advantage with this notation is, to perform logical operations we can use the logical operations available in IBM 1800's Assembly language. This facility results in considerable saving in computation time.

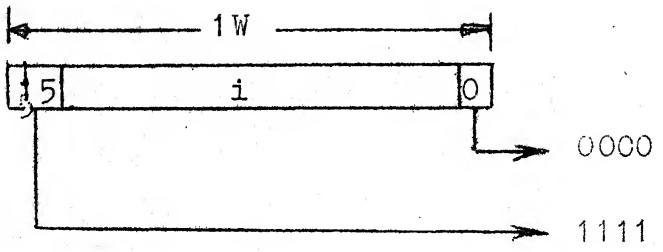

Fig. 3.4 Format of FUNC

The format of BLDF is also similar to that of FUNC. For each line FUNC and BLDF have 16 words. Again the key to retrieve any value is the line number.

(c) PATH: The format of PATH is shown in Fig. 3.5. It is a two dimensional array. Each row in PATH corresponds

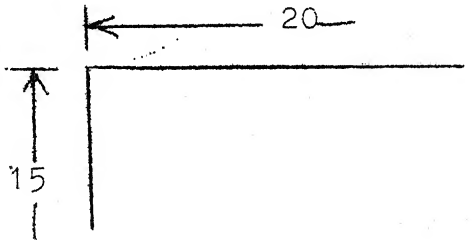

Fig. 3.5 Format of PATH

to a path in the circuit. The first column in any row contains the initial node (line) of the path and the last entry in any row is the POL. Associated with the PATH is a column containing flags to indicate whether each path has a RFO line in it or not.

(d) STACK: The stack called STACK contains 20 rows and 2 columns. SP is the stack pointer. This is used as a storage during the enumeration of paths.

(e) FT: The FT contains the diagnostic table. It carries the input tests, the test matrix and the fault or the first entry of a fault equivalent class. Associated with each fault is a LINK1 which points to the next line of the fault equivalence class in the ECT. The format is given in Fig. 3.6.

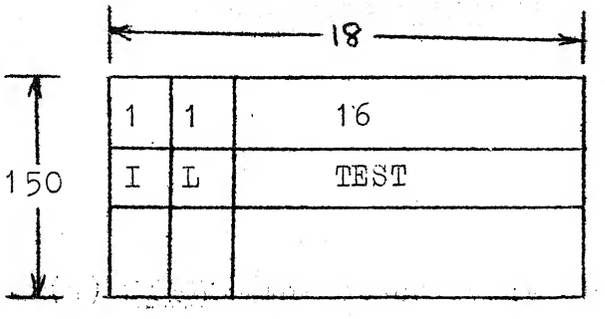

Fig. 3.6

I : Fault I.

L : LINK1 pointing to ECT or 0.

TEST:The test set for fault I.

Since each line can have 2 faults, we need to code the faults. It is done as follows:

Let NG be the no. of lines in the UUT. Then,

(i) The s-a-0 fault in line I is represented by I itself.

(ii) The s-a-1 fault in line I is represented by I+NG.

Given a fault J, it is uniquely possible to find out the line and the type of fault by knowing NG.

(i) If $J \leqslant NG$, then the line is J and the fault type is s-a-0.

(ii) If $J > NG$, then the line is J-NG and the fault type is s-a-1.

(f) ECT: It is not necessary to accommodate all the entries corresponding to a fault-equivalence class in FT. This is done separately in ECT. ECT has a list structure. LINK1 of FT points to an element of ECT. Each record of ECT has two fields: one the fault and the other, the pointer to the next element in ECT. Fig. 3.7 gives the structure.

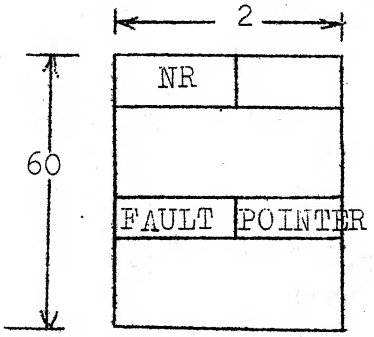

Fig. 3.7 ECT Structure

NR: no. of records in ECT.

POINTER points to another record of ECT or is zero.

3.3.5 Module 1: Fig. 3.8 gives the flow chart of MOD 1. The input data cards are read. CKT data base is built using suitable packing of inputs. Once the end of file card is encountered control is transferred to Module 2.

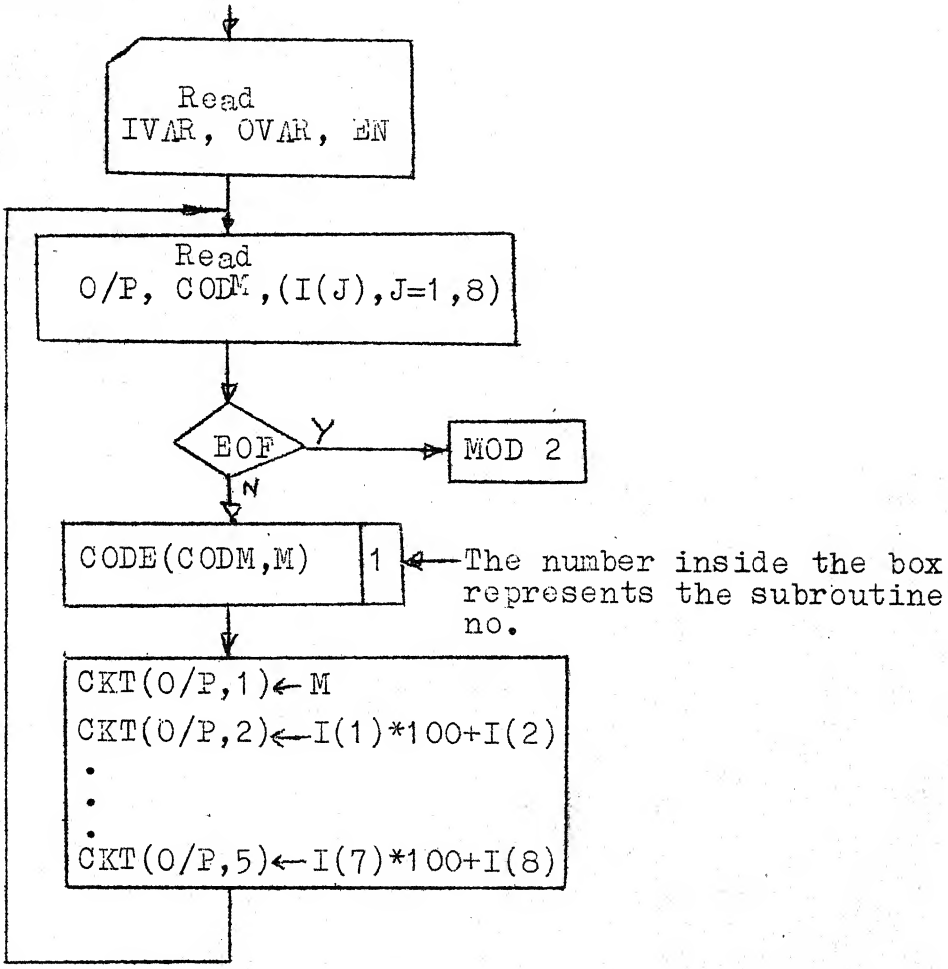

Fig. 3.8 F/C of Module 1

1) Subroutine CODE(CODM,M)*: This subroutine finds the internal code M of the gate whose alphanumeric code is CODM.

3.3.6 Module 2: The operations performed in this module are:

1) To initialize the logical values of PILs.

2) To evaluate the logical values of lines starting from the lowest line to the highest line number.

The simulation of the circuit is done with this module. The simulation is a gate level simulation. The flow chart is given in Fig. 3.9.

2) STRT(FUNC(J),I,J)*: The primary input lines are to be set to the suitable logical values. This routine performs that function. The values are stored in the routine through a data statement and is accessed using I. The format of data structure is such that not much data is to be stored in the routine. Logical values which need length less than single computer word, can be obtained by trimming the excess bit locations.

3) GAND(FUNC(I),FUNC(I))*: This routine is written in IBM 1800 Assembly language. This takes less storage space and is also fast compared to a routine written in FORTRAN to do the same function. The function performed by this sub-program is:

FUNC(I) $\leftarrow$ FUNC(I).FUNC(J).

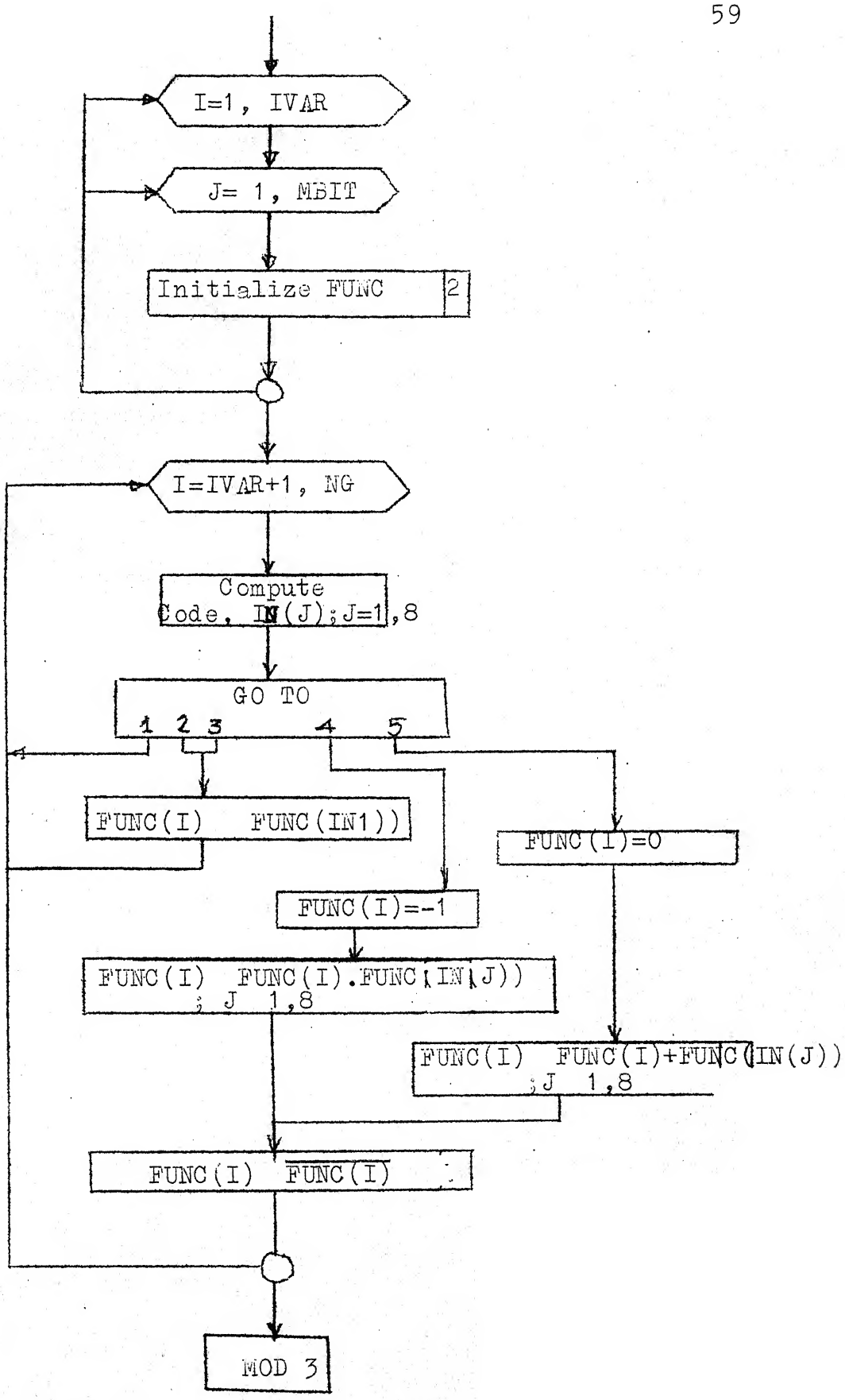

Fig. 3.9 Module 2

4) GOR (FUNC(J),FUNC(I))*: This is similar to the GAND routine but does the following:

$$FUNC(I) \leftarrow FUNC(I) + FUNC(J)$$

5) COMP(FUNC(I))*: This routine complements the function by doing:

$$FUNC(I) \leftarrow \overline{FUNC(I)}$$

3.3.7 Module 3: Since the UUT may have more than one POL, some line in the UUT may affect some POL and others may not. Similarly some fan-out lines with respect to a POL may in fact, be RFO lines for other POLs. It is essential to perform the operations with respect to every POL. The functions accomplished by Module 3 are:

1. To initialize PATH, FAN, PTYPE
2. To enumerate the paths
3. To rearrange the paths
4. To determine RFO lines,

To enumerate the paths, a stack called STACK is used. Starting from the POL we climb down via a branch to a lower level and try to reach a PIL. Once we have reached, we collect the lines through which we have traced the path and transfer them into PATH. Then we back track one step and create a branch through another PIL. If in the backtracking

process we come back to our starting point, then we would have enumerated all the paths starting from the PILs ending with the POLs.

To determine the RFO lines we make a comparison of those paths which have the same starting line. We intersect these two paths. If the intersection is empty, we repeat the process with other paths in the PATH till we exhaust all the paths. If the intersection is not empty, we find out the last entry in the intersection. This line is the RFO line. We set the PTYPE corresponding to these paths to indicate that these paths contain a RFO line.

6) PUT(I,FAN)*: I is the line that is to be stored in FAN. This routine checks FAN whether it contains I. If it does not contain I is included in FAN and the control returns.

3.3.8 Module 4: The BD of the POL with respect to the lines in RFO is evaluated in this module. There are no straight forward methods of achieving this. The logical value of the RFO line is set to 0 first. The logical values of those lines which are independent of the RFO line are transferred from FUNC to the working storage. Using a procedure similar to that of Module 2, the logical values of lines at a higher level than the RFO line are evaluated. We continue this process till we reach the POL. Now the value of the POL

is that corresponding to the RFO line being zero. The same procedure is repeated with the RFO line being set to 1. The logical values of POL corresponding to RFO line being 0 and 1, are operated using an EX'OR. The result is the BD of the RFO line. The following operations are performed.

1. The lines in PATH are partitioned into two classes; one containing lines which lie at a higher level than the RFO line; the other containing those lines which are independent of the RFO line.

2. The BD is evaluated as described earlier.

7) PUTIN(STOR1,I,SWCH)*: The element I is reached in STOR1. If found SWCH is reset. Otherwise I is entered in STOR1.

8) ASEND(STOR1)*: Before evaluating the logical function of a line, it is necessary to evaluate the logical functions of all the lines below it. The only way of ensuring this is to arrange all the lines in ascending order in a store and to start executing from the smallest line to the higher one. ASEND arranges the elements of STORS in ascending order and shifts the whole vector to the left by suppressing the left most zero elements in STOR1.

9) GEOR(FUNC1,FUNC(2))*: The operation performed by GEOR is:

FUNC2 $\leftarrow$ FUNC1 $\oplus$ FUNC2.

3.3.9 Module 5: Boolean difference chain evaluation is implemented in this module. The BDs of the lines in PATH other than those in RFO are evaluated. The flow chart for this module is given in Fig. 3.10.

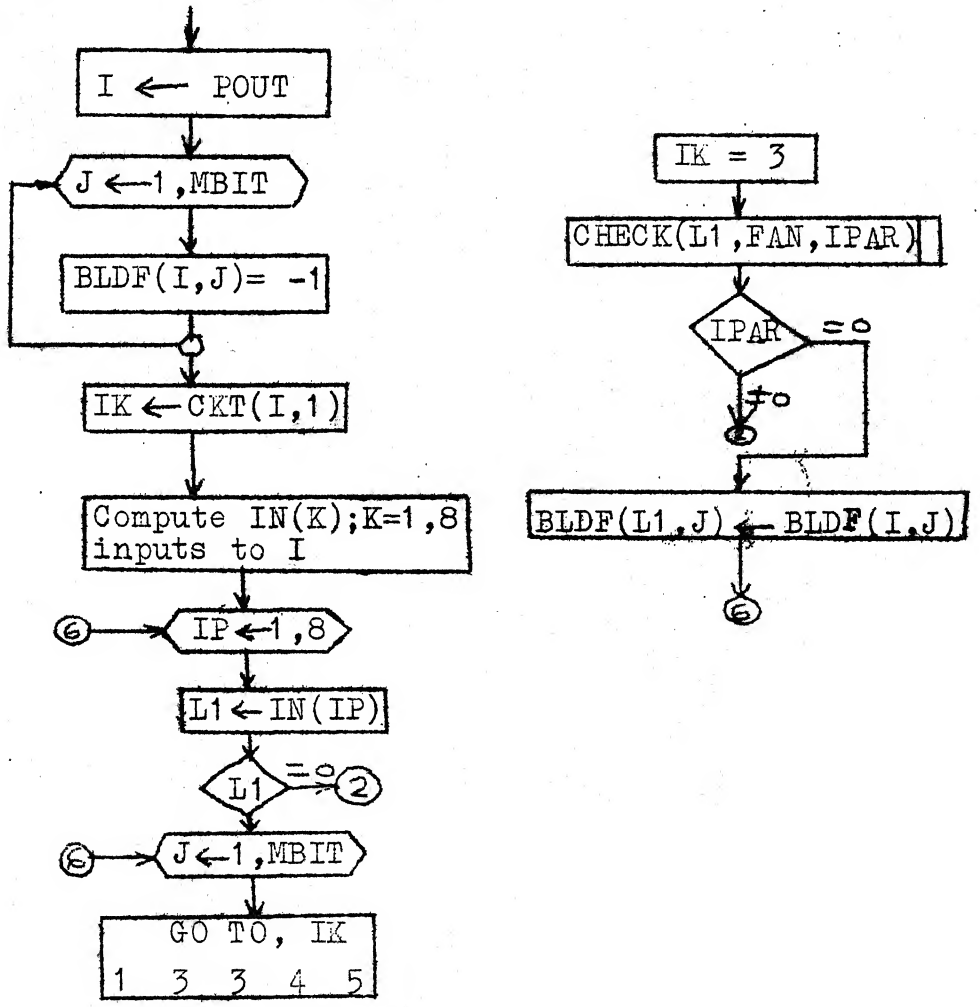

Fig. 3.10

10) CHECK(I,FAN,IPAR)*: This subroutine checks if line I is a member of RFO lines. IPAR is set if it is so, else IPAR is reset.

1). Module 5 separates the non RFO lines and stores them in STOR1 and STOR2 concatenated together. It arranges the elements in ascending order and the BD evaluation is carried out as shown in the flow chart.

3.3.10 Module 6: The operations performed in Module 6 are:

1). Determining all the lines in the circuit with respect to the POL.

2). For each gate, finding out the faults of the input and output lines of the gate and determining the fault-equivalence class of the gate.

3). Preparing FT and ECT data bases for use in Module 7.

11) INCR(I,J,N,M)*: The following steps are performed:

1. J ← J+M;
2. If J ⩽ N continue; else go to 5
3. I ← I+1 ;
4. J ← J-N;
5. return;

12) GET(I,J,K)*: This subroutine fetches the last entry of a fault-equivalence class is the record number of the first

entry of the FEC in FT is specified. I is the record number of FT. The LINK1 in FT is obtained. Using LINK1, the appropriate record in ECT is retrieved. If the LINK or POINTER associated with the retrieved record is zero, the values are returned. Else a new record is fetched and the operations are repeated till we end up with the last record of a list. I is the last record number and K is the entry.

3.3.11 Module 7: The following steps are carried out in Module 7:

1. Computing the $\not{f}^{fault}$ test using FUNC & BLDF
2. Minimising the FT
3. Rearranging the FT
4. Further minimisation
5. Outputing the FT.

The above tasks are accomplished through a set of subroutines.

13) FALT(NFIL1,NFIL2)*: NFIL1, NFIL2 are two system parameters. For each record in FT, the line number I and the type of fault f are computed from the fault entry of the record.

1. FUNC(I) and BLDF(I) are fetched.
2. If f = 0; Test $\leftarrow$ FUNC(I).BLDF(I).
3. If f = 1; Test $\leftarrow$ $\overline{\text{FUNC(I)}}$.BLDF(I).

14) MINIM*: The minimisation of the FT is done by MINIM. Each test vector is compared with the other. If they are equal, the fault corresponding to the second is included with the fault of the first.

15) COMPR(FUNC1, FUNC2,I,J)*: FUNC1, FUNC2 are two vectors. They are compared element by element for I elements. If they match J is set to 1, otherwise to zero.

16) TRIM(LBIT,I)*: The number I is trimmed of its leading LBIT bits.

3.3.12 Implementation: The implementation of SYSTEM I is successfully made in IBM-1800 computer. The available core memory of the computer for user is 8K words. If SYSTEM1 is implemented as it is, it takes more than 12K words. The whole program is segmented into 3 different parts. The data bases FUNC, BLDF, FT are transferred via IBM-1810 disk. Other important data bases and variables are transferred via the common block of the memory. The individual segments, however, can be run continuously once the first segment is started. This is done by calling LINK (Segment) routines. The three segments are stored in the fixed area of 1800 computer and can be accessed at a fast rate to the memory. The data bases are created as files in disk. The file system used is a simple one as permitted by FORTRAN.

3.4 Design of SYSTEM II:

The functions performed by SYSTEM II are:

1. Reading the FT (from cards)
2. Constructing the data base FT (in core)
3. Applying the tests to the UUT and observing the responses.
4. Diagnosing the results.

The flow chart is given in Fig. 3.11. The data structure of the two data bases FT and FECT are given in Figs 3.12 and 3.13.

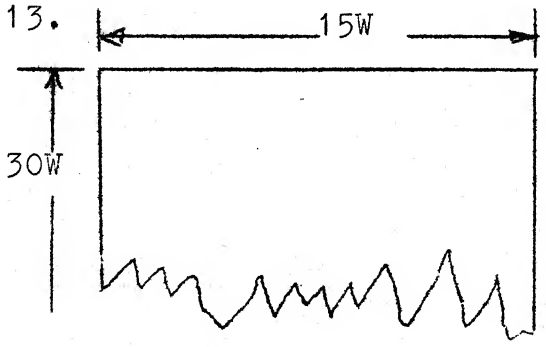

Fig. 3.12 Format of FECT

The FECT contains the fault-equivalence classes. Each row contains an equivalent class. There can be at most 15 faults in an equivalence class.

The faults that are detected by the system are packed into 5 words and stored in FT. A 1 in a row indicates that the test associated with that row can detect the

fault associated with the column. The bit positions left to right in FT correspond to faults in FECT top to bottom.

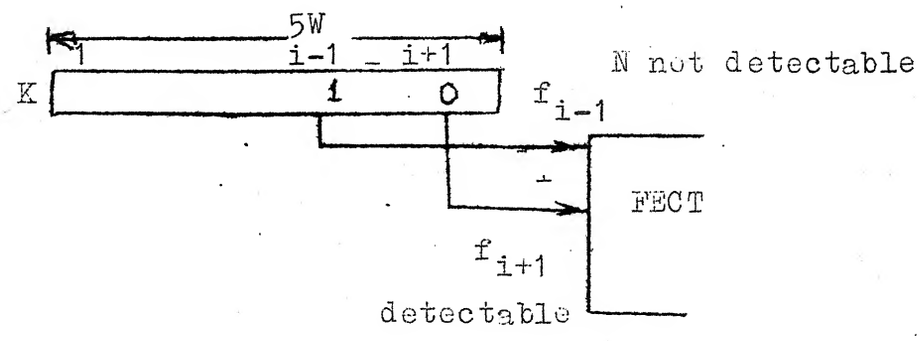

Fig. 3.13 Format of FT

RITE (FUNC,IFUNC)*: FUNC and IFUNC are vectors of dimension 70 and 5 respectively. 16 elements of FUNC are packed into a word of IFUNC.

FIND(IFUNC,M,K)*: This subroutine locates a one's position in IFUNC and that value is stored in M. Simultaneously it searches for another 1 in IFUNC. If the routine finds another 1, K is set to 1 and the subroutine returns control.

The subroutines developed in SYSTEM I can be used for SYSTEM II also. I/O subroutines for digital direct input and output should be developed. Because the ATE H/W is not available, the SYSTEM II is not implemented on IBM-1800. However, it can be done easily.

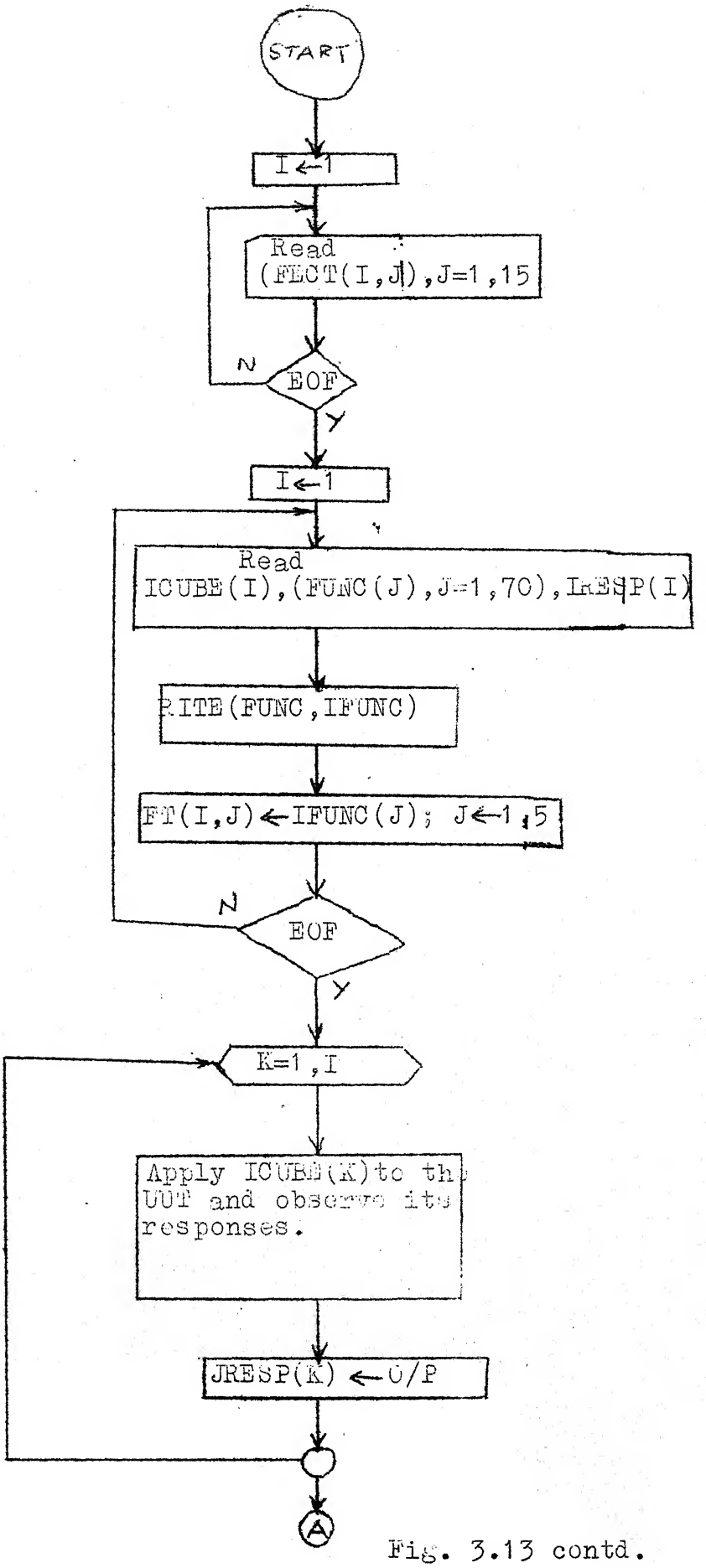

Fig. 3.13 contd.

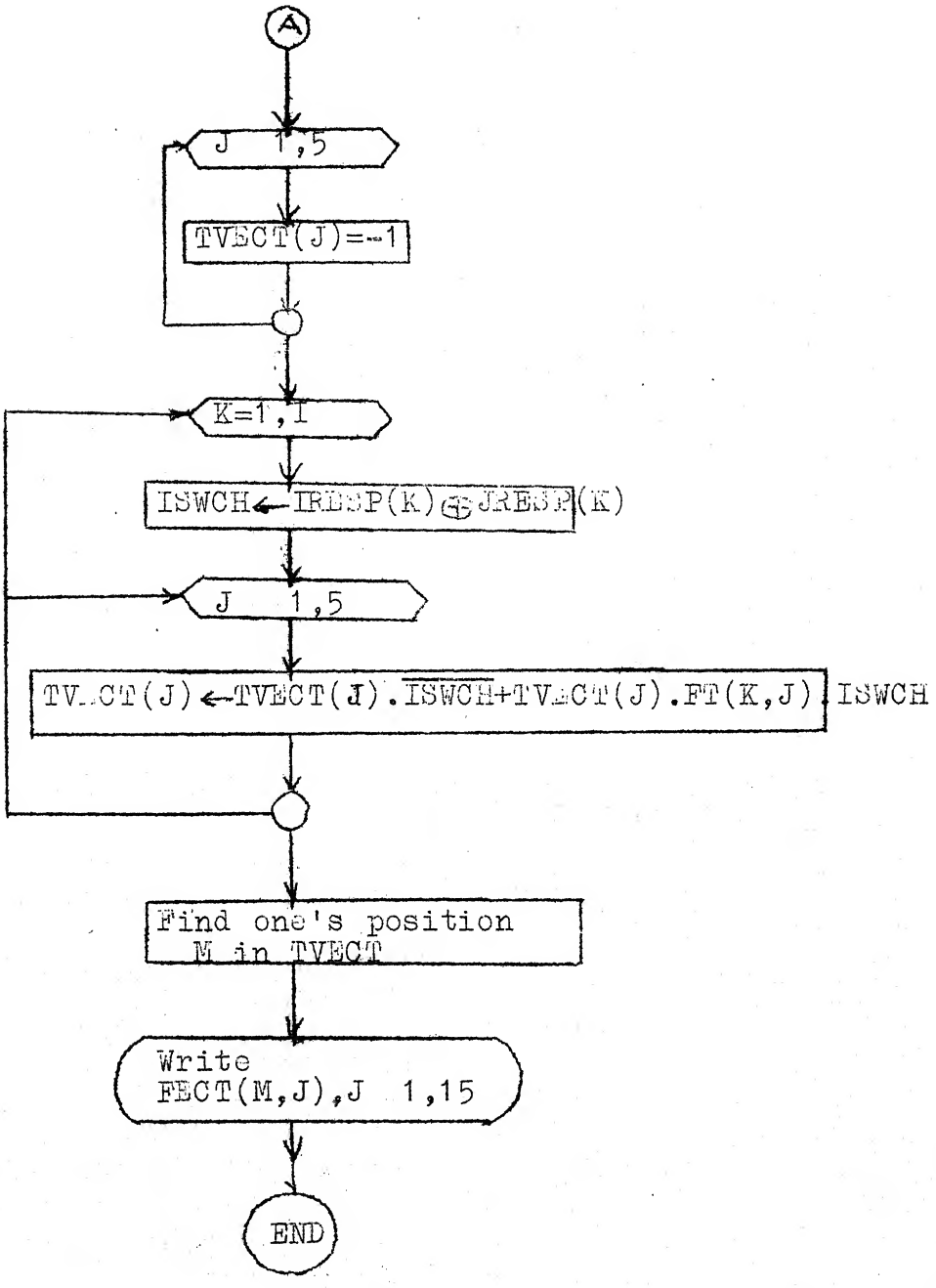
A
J 1,5
TVECT(J)=-1
K=1,I
ISWCH←IRESP(K)⊕JRESP(K)
J 1,5
TVECT(J)←TVECT(J).ISWCH+TVECT(J).FT(K,J).ISWCH
Find one's position
M in TVECT
Write
FECT(M,J),J 1,15
END

Fig. 3.13

CHAPTER 4

RESULTS, CONCLUSIONS AND SUGGESTIONS

4.1 Results of the Implementation:

SYSTEM I is implemented in IBM-1800 successfully. The diagnostic tables generated are given in Appendix, for the examples run on the system. The average time taken for processing a circuit having 5-10 gates is 3 secs. A comparison of SYSTEM I can be made with other systems implemented [24]

Table 4.1

Variables	OTHER SYSTEM[1]					SYSTEM I
	1	2	3	4	5	
1. No. of Inputs/Outputs	3/1	4/1	15/1	11/4	55/7	8/8
2. No. of Basic Gates	5	8	21	62	95	99
3. Core store used	9K	9K	12K	14K	26K	9K
4. Time to Generate Tests (in secs)	1	1	17	25	252	3[2]
5. Time to calculate the O/P associated with each Test(in secs)	1	1	2	3	58	-[3]

1. System implemented by Bennetts
2. Computing Time for circuits with small no. of gates.
3. This time is included in the time to generate the tests.

The following conclusions can be drawn from the comparison.

1) For circuits with PILs, SYSTEM I is efficient for it takes less memory compared to the 3rd example run on the other system. When the no. of PILs are increased, the memory required increases at a fast rate in SYSTEM I. This limitation can be attributed to the data structure. For circuits with small number of PILs, it is suitable. When the PILs are increased, the memory increases as $\lceil 2^r/16 \rceil$ words where r is the number of PILs.

2) There is no restriction of the number of POLs imposed by the core except that the computing time may be Large.

3) In other systems the details about the RFO lines are to be provided with the input data. SYSTEM I has an imbedded module which determines all the RFO lines with respect to a POL.

4) There is no separate procedure required to compute the correct output response associated with a test. It is evaluated during the normal execution of the program.

4.2 Suggestions for Further Work:

SYSTEM I lacks many features which can be implemented to enhance its capacity.

1. The study of better data structures to represent FUNC and BLDF, is a necessity because most of the digital circuits have atleast 16, PILs. So a test generation system should be capable of accepting circuits of that size.

2. Digital circuits are becoming bigger in size by having the basic building blocks quite complex. SYSTEM I wants all the basic components to be NANDs or NORs. A macro processor in similar lines to that described in [23] can be implemented. It should be possible to represent all the basic building blocks of a circuit as macros with internal components of the macros being NANDs or NORs. Once the macros are provided to the system, references to the macros can be made.

3. SYSTEM II can be implemented on IBM-1800. The successful operation of SYSTEM II lies with the ATE H/W to be attached to the IBM-1800 digital I/O terminals.

4. The thesis deals with the fault diagnosis techniques for combinational circuits only. But most of the digital systems are infact sequential. Eventhough the procedures to detect faults in sequential circuits are complex, an actual system to diagnose the faults in such circuits will be a boon to the users of Automatic Test Equipments.

REFERENCES

[1] Friedman, A.D., and Menon, P.R., 'Fault Detection in Digital circuits', Prentice-Hall Inc., New Jersey, 1971.

[2] Kohavi, Z., 'Switching Theory and Finite Automata ', McGraw-Hill Book Co., New York, 1970.

[3] Sellers, F.F., Hsiao, M.Y., Bearnson, L.W., 'Error Detecting Logic for Digital Computers', Mc-Graw Hill Book Co., New York, 1968.

[4] To, K., Tulloss, R.E., 'Automatic Test Systems', IEEE Spectrum, Sept. 1974.

[5] Chang, H.Y., etc., 'Fault Diagnosis of Digital Systems'. Wiley-Interscience, New York, 1970.

[6] Manning, E.G., and Chang , H.Y., 'A Comparison of Fault Simulation Methods for Digital Systems', Digest of the First Ann. IEEE Comp. Conf., 1967.

[7] Roth, J.P., 'Diagnosis of Automata Failures: a Calculus and a Method', IBM Journal of R&D, vol. 10, pp 278-291, 1966.

[8] Sellers, F.F., Hsiao, M.Y., and Bearnson, L.W., 'Analysis of errors with the Boolean Difference', IEEE Trans. on Computers, C-17, pp 676-683, 1968.

[9] Hsiao, M.Y., and Chia, D.K., 'Boolean Difference for Test Pattern Generation, Part I - Combinational Circuits' Proc. 4th Annual Princeton Conf. Inform. Science, Princeton, New Jersey, Mar. 20, 1970.

[10] Hayes, J.P., 'Testing Logic circuits by Transition Counting', Proc. of 5th International Symposium on Fault Tolerant Computing, Paris, June, 1975.

[11] Thayse, A., and Davio, M., 'Boolean Differential Calculus and its Application to Switching Theory', IEEE Trans. on Computers, vol. C-22, pp. 409-420, Aprl. 1973.

[12] Akers, S.B., 'On a Theory of Boolean Functions', Journal of SIAM, vol. 7, pp. 487-498, Dec. 1959.

[13] Yau, S.S., and Tang, Y.S., 'An Efficient Algorithm for Generating Complete Test Sets for Combinational Logic Circuits', IEEE Trans. on Computers, vol C-20, Nov. 1971.

[14] Reed, I.S., 'A Class of Multiple-error-correcting Codes and the Decoding Scheme, Trans. Inst. Radio Engrs., IT-4, pp 38-49, 1954.

[15] Armstrong, D.B., 'On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic circuits', IEEE Trans. on Electronic Computers, vol. EC-15, Feb. 1966.

[16] Chang, H.Y., 'An Algorithm for Selecting an Optimum Set of Diagnostic Tests', IEEE Trans. Electronic Computers, vol. EC-14, Oct. 1965.

[17] Scheff, B., 'Using a DT Structure as the Input Language Format for Programming Automatic Test Equipment Systems', IEEE Trans. Electronic Computers vol. EC-14, April 1965.

[18] Seshagiri, N., 'A Decision Table Approach to Self-Diagnostic Computers', Proc. of IEEE, Dec. 1967.

[19] Rajaraman, V., 'Design of Information Systems', Lecture Notes, Indian Institute of Technology, Kanpur, Feb. 1975.

[20] Kautz, W.H., 'Fault Testing and Diagnosis in Combinational Digital circuits', IEEE Trans. on Computers, vol. C-17, No. 4, April 1968.

[21] Price, W.T., 'Elements of IBM 1130 Programming', Holt, Rinehart & Winston, New York, 1968.

[22] 'IBM 1800 MPX Assembly Programming Manual'.

[23] Donovan, J.J., 'Systems Programming', McGraw-Hill Inc., New York, 1972.

[24] Bennetts, R.G., 'A Realistic Approach to,detection Test Set Generations for Combinational Logic Circuits', The Computer Journal, vol. 15, No. 3, Aug. 1972.

[25] Judd, D.R., 'Use of Files', Macdonald, London, 1973.